Proposal

VideO Processing Language

（VOPL）

Sep. 20th, 2008

Baolin Shao （ bs2530 ）

Huning Dai (hd2210)

Jia Li （ jl3272 ）

Xuyang Shi （xs2137 ）

Introduction

Thanks to the information technology, now we are living a digital life, and gone are the days, when people must have their pictures developed and put into albums. Instead, we began to use digital cameras and to save pictures in a computer years ago, which led to an increasing popularity of picture-editing tools. Currently, however, booming of multi-media requires us move one-step further: we not only want to easily process images but also to manipulate videos. Unfortunately, there are few languages that can help programmers effectively develop video-editing tools, keeping them away from miscellaneous video processing routines. For example, C and C++ are effective, but they require experienced programmers to do everything from scratch, such as designing a video class or carefully calculating pixels' positions. Compared with C and C++, Matlab is more productive, because it provides mechanisms to directly manipulate images so that complicated image processing becomes simply calling Matlab's build-in functions. It is, however, still impossible for programmers to directly work on videos. Therefore, we propose to develop a VideO Processing Language (VOPL) by which programmers can easily edit videos. What makes VOPL unique is that it facilitates video programming by incorporating English-like statements into the language, much as SQL does for databases. Besides, VOPL provides basic sequential and flow-control operations with a C-like semantics.

Scope and Limitations

VOPL has basic arithmetic operations and flow-control constructs with a C-like semantics. This basically allows programmers to specify any mathematical algorithm they apply to a video. VOPL also offers common routines in video editing, such as open, delete, insert, etc. As a special feature in VOPL, these instructions are all English-like statements. What is more, VOPL provides an editing mechanism for programmers so that they can easily apply customized operations to a video.

Although VOPL seems to be powerful enough to make "stupid video programming" possible, it still has many limitations. Primarily, there is a constraint on the video's format that VOPL can handle. As a course project, VOPL only accepts ".yuv" format,

which contains only raw data without any compression. For more complex file format, such as rmvb, mpeg and H.26x, we can extend VOPL by adding libraries. The second issue not addressed in VOPL is efficiency. We assume that computers are fast enough and have enough resources to complete a task, which is based on the following arguments. Firstly, video processing is always a resource-consuming task even implemented by C. Secondly, since computers are growing faster and faster, the first issues should be considered less and less. Finally, and most importantly, simplicity and productivity should have first priority among any other considerations, because people, not computers, are the most expensive resource.

Language Overview

In this section, first we will introduce VOPL's key constructs like load, store, insert, delete, copy and update. Afterwards, we will present a small VOPL program, which loads two videos, smoothes part of the first one, and then inserts it into some specific place of the second one.

Load

Syntax

load *video1* ***from*** *file1* ***with*** *w* ***and*** *h;*

Semantics

Video1 must be a variable defined with a build-in type *video* and *file1* is a string, indicating the path of the file. *"w"* and *"h"* correspond to the width and height of the video respectively. The load instruction will put data into the video1 variable. This instruction frees programmers from worrying how data are organized in a file.

Store

Syntax

store *video1* ***to*** *file2;*

Semantics

As *load*, *store* also requires *video1* and *file2* to be a *video* type and a string respectively. As its name tells, store will put the processed data back to *file2*.

Insert

Syntax

insert *video1* ***into*** *video2* ***from*** *i;*

Semantics

Video1 and *video2* are two *video* variables and *i* is an integer number. This instruction will insert a video, say *video1*, into another video, say *video2*, from the i-th frame.

Delete

Syntax

delete *video1* ***from*** *i1* ***to*** *i2;*

Semantics

Delete will cut off *video1* from the *i1-th* frame to the *i2-th* frame and combine the remaining two parts together.

Copy

Syntax

copy video1 **from** i1 **to** i2 **into** video2;

Semantics

Copy will duplicate video1's frames from i1 to i2 and put them into video2.

Update

Syntax

update *video1* ***from*** *i1* ***to*** *i2*

{ statements };

Semantics

Update is the most powerful and complex instruction in VOPL. It allows programmers to design any algorithm and iteratively apply it to a sequence of frames. *Statements* within this block can be any C-like statement, but cannot be video related statements, such as *load, store, copy, insert, delete* and *update*. Function calls are prohibited either. To symbolically manipulate the frame in each iteration, VOPL provides a dummy frame variable, called *this*. Programmers can access pixels of this dummy frame by *this(x,y)*, where x and y are positions of this pixel.

Sample program

```
void main()
{  video v1,v2;
   load v1 from "/film1.yuv" with 128 and 128;
   load v2 from "/film2.yuv" with 128 and 128;
   update v1 from 5 to 10
   {// a very simple and stupid smoothing algorithm
     int i,j;
     for(i=0;i<128-1;i++)
          for(j=0;j<128-1;j++)
          this(i,j)=(this(i,j)+this(i+1,j)+this(i,j+1)+this(i+1,j+1))/4;
   };
   delete v1 from 0 to 4;
   delete v1 from 11 to end;
   insert v1 to v2 from 15;
   store v1 to "/film1_new.yuv";
   store v2 to "/film2_new.yuv";
 }
```

Project Plan

Milestones and Timeline

S#	Milestones	Deliverables	Milestone Date	Responsibility
01	Completion of Project plan	Project Proposal	2008-9-24	PM
02	Completion of Language Reference Manual	Language Reference Manual	2008-10-10	Everyone
03	Completion of Coding + Unit Testing	Code, Unit test results	2008-11-23	Everyone
04	Completion of Integration testing	Code, Integration test results	2008-12-1	ML, PM
05	Completion of Presentation	Presentation of the language	2008-12-6	PM
06	Completion of Final Report	Final Project Report	2008-12-15	Everyone

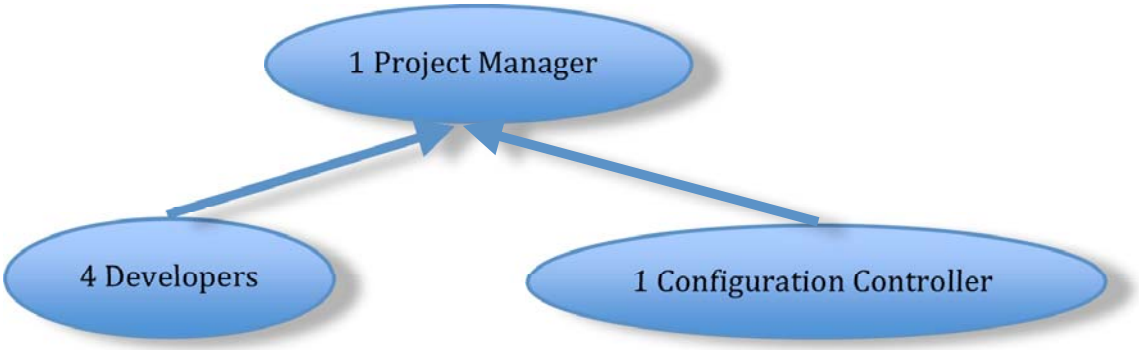
Life Cycle Process

LC Stage	Deliverables
Plan	Project plan
Requirement Analysis	Language Manual
High Level Design	HLD Integrated Test Plan
Detailed Level Design	DLD Unit Test Plan
Coding	Source Code, Unit test results
Testing	Test results
Deployment	Product, Presentation, Final Report

Reviews

LC Stage	Review Item	Type of Review Group Review/One Person Review etc
Plan	PM plan	Group review
Requirement Analysis	Language Manual	Group review
High level design	HLD, Integrated Test Plan	Group review
Detailed level design	DDL, Unit Test Plan	Single / Peer review
Coding	Source Code, test results	Single / Peer review
Final Report	Final Report	Group review
Presentation	Presentation	Group review

Project Organization



Roles and Responsibilities

Role	Responsibilities
Project Manager	Interaction with quality person Responsible for the project delivery and entire team coordination Responsible for handing over the project Creation of PM plan along with team members
Configuration Coordinator	Maintain CVS Maintain all versions of coding Manage the file ‘defect tracking’ everyday Make sure the defect are dealt by the owner
Developer	Coding Participation in the team Can give suggestions Testing of self coded modules and also other modules